biPCPG

Release 0.1.0

Carlos Saenz de Pipaon

Mar 24, 2022

CONTENTS:

CHAPTER ONE

BIPCPG

This package implements the Bipartite PCPG (biPCPG) algorithm[1], a generalisation of the Partial Correlation Planar Graph (PCPG) algorithm[2]. The PCPG is a correlation-filtering method for the construction of networks intended for use on multivariate time series datasets with a single sample. The biPCPG framework generalises this approach to allows its use on similar datasets containing multi-sample multivariate time series.

The biPCPG package offers three main tools:

- Handling the dataset, via the `reshape_year_matrices_to_time_series_matrices()` function.
- Applying the PCPG, via the `PCPG` class.
- Performing a bootstrap on the PCPG network's edges, via the `get_bootstrap_values()` function.

We recommend having a look at the tutorial to get started.

1.1 References

[1] Saenz de Pipaon Perez C, Zaccaria A, Di Matteo T. Asymmetric Relatedness from Partial Correlation. Entropy. 2022; 24(3):365. <https://doi.org/10.3390/e24030365>

[2] Kenett DY, Tumminello M, Madi A, Gur-Gershgoren G, Mantegna RN, Ben-Jacob E (2010) Dominating Clasp of the Financial Sector Revealed by Partial Correlation Analysis of the Stock Market. PLoS ONE 5(12): e15032. <https://doi.org/10.1371/journal.pone.0015032>

CHAPTER

TWO

INSTALLATION

In order to install the bipcpg package clone (or download and unpack) the latest version from Github. From the folder containing biPCPG's setup.py file run:

```
pip install .
```

For example:

```
git clone https://github.com/cspipaon/biPCPG.git
cd biPCPG
pip install .
```

To install in an Anaconda virtual environment (recommended) with the required packages:

```
git clone https://github.com/cspipaon/biPCPG.git
cd biPCPG
conda create --name <env_name> python=3.8 --file requirements.txt -c conda-forge
conda activate <env_name>
pip install .
```

where <env_name> should be replaced by the desired name of the virtual environment.

CHAPTER

THREE

TUTORIAL

The `bipcpg` package facilitates the computation of a Partial Correlation Planar Graph (PCPG) network for datasets with a bipartite structure, as well as the preparation of the data for this purpose and a bootstrapping procedure to assess the reliability of the edges in the network. Below we give an example of how to apply these methods to a toy dataset consisting of countries and the products they export with the aim of obtaining a PCPG network with products as nodes.

3.1 Dataset structure

Consider a bipartite dataset containing the quantity (in millions of dollars) of a set products exported by a set of countries. In this toy example, assume we have data for 4 countries, 4 products over a 5 year span with one data point per year. Lets denote the countries c_1 to c_4, the products p_1 to p_4 and the years y_1 to y_5. Furthermore, denote the quantity of exports a given country does of a given product in a given year by e^{y}_{cp}.

This sort of dataset is usually distributed as a collection of tables indexed over time containing the data for that given year. Following our example, we would have the following tables or matrices.

For the first year y_1:

	p_1	p_2	p_3	p_4
c_1	$e^{y_1}_{c_1p_1}$	$e^{y_1}_{c_1p_2}$	$e^{y_1}_{c_1p_3}$	$e^{y_1}_{c_1p_4}$
c_2	$e^{y_1}_{c_2p_1}$	$e^{y_1}_{c_2p_2}$	$e^{y_1}_{c_2p_3}$	$e^{y_1}_{c_2p_4}$
c_3	$e^{y_1}_{c_3p_1}$	$e^{y_1}_{c_3p_2}$	$e^{y_1}_{c_3p_3}$	$e^{y_1}_{c_3p_4}$
c_4	$e^{y_1}_{c_4p_1}$	$e^{y_1}_{c_4p_2}$	$e^{y_1}_{c_4p_3}$	$e^{y_1}_{c_4p_4}$

for the second year y_2:

	p_1	p_2	p_3	p_4
c_1	$e^{y_2}_{c_1p_1}$	$e^{y_2}_{c_1p_2}$	$e^{y_2}_{c_1p_3}$	$e^{y_2}_{c_1p_4}$
c_2	$e^{y_2}_{c_2p_1}$	$e^{y_2}_{c_2p_2}$	$e^{y_2}_{c_2p_3}$	$e^{y_2}_{c_2p_4}$
c_3	$e^{y_2}_{c_3p_1}$	$e^{y_2}_{c_3p_2}$	$e^{y_2}_{c_3p_3}$	$e^{y_2}_{c_3p_4}$
c_4	$e^{y_2}_{c_4p_1}$	$e^{y_2}_{c_4p_2}$	$e^{y_2}_{c_4p_3}$	$e^{y_2}_{c_4p_4}$

and similarly for years y_3, y_4 and y_5.

In order to use such a dataset with the `bipcpg` package, we have to reshape the data such that, instead of having a matrix per time index, we have a matrix per element of one of the two sets of variables. These matrices should have rows representing time indices and columns representing the complementary set of variables. In our example, instead of a matrix per year, we could reshape the dataset into either a matrix per country or a matrix per product. If we shape the data such that we have one matrix per country and apply the Bipartite PCPG (biPCPG) algorithm, we would obtain a network whose nodes are products, and vice versa.

Given we want to obtain a network of products, we need to reshape our data such that we have four matrices, one per country, containing the export time series for each products as columns. Using the notation introduced above, these matrices have the following structure:

The matrix for country c_1:

	p_1	p_2	p_3	p_4
y_1	$e^{y_1}_{c_1p_1}$	$e^{y_1}_{c_1p_2}$	$e^{y_1}_{c_1p_3}$	$e^{y_1}_{c_1p_4}$
y_2	$e^{y_2}_{c_1p_1}$	$e^{y_2}_{c_1p_2}$	$e^{y_2}_{c_1p_3}$	$e^{y_2}_{c_1p_4}$
y_3	$e^{y_3}_{c_1p_1}$	$e^{y_3}_{c_1p_2}$	$e^{y_3}_{c_1p_3}$	$e^{y_3}_{c_1p_4}$
y_4	$e^{y_4}_{c_1p_1}$	$e^{y_4}_{c_1p_2}$	$e^{y_4}_{c_1p_3}$	$e^{y_4}_{c_1p_4}$
y_5	$e^{y_5}_{c_1p_1}$	$e^{y_5}_{c_1p_2}$	$e^{y_5}_{c_1p_3}$	$e^{y_5}_{c_1p_4}$

the matrix for country c_2:

	p_1	p_2	p_3	p_4
y_1	$e^{y_1}_{c_2p_1}$	$e^{y_1}_{c_2p_2}$	$e^{y_1}_{c_2p_3}$	$e^{y_1}_{c_2p_4}$
y_2	$e^{y_2}_{c_2p_1}$	$e^{y_2}_{c_2p_2}$	$e^{y_2}_{c_2p_3}$	$e^{y_2}_{c_2p_4}$
y_3	$e^{y_3}_{c_2p_1}$	$e^{y_3}_{c_2p_2}$	$e^{y_3}_{c_2p_3}$	$e^{y_3}_{c_2p_4}$
y_4	$e^{y_4}_{c_2p_1}$	$e^{y_4}_{c_2p_2}$	$e^{y_4}_{c_2p_3}$	$e^{y_4}_{c_2p_4}$
y_5	$e^{y_5}_{c_2p_1}$	$e^{y_5}_{c_2p_2}$	$e^{y_5}_{c_2p_3}$	$e^{y_5}_{c_2p_4}$

and similarly for countries c_3 and c_4.

Now lets see how the above translates into code. Take the following `dataset`, with a matrix **per year** as an example:

```
>>> import numpy as np
... dataset = [np.array([[1.2, 3., 1., 5.4],                  # y_1 data
...                      [10.2, 8.8, 11.7, 15.2],             #
...                      [101.7, 99.7, 104.2, 103.8],         #
...                      [1001.9, 1002.7, 1000.7, 1004.7]]),  #
...            np.array([[0.1, 5.2, 4.5, 4.2],                ## y_2 data
...                      [9.1, 12.2, 13.4, 11.7],             ##
...                      [105.5, 102.9, 106.5, 101.9],        ##
...                      [1004.4, 999.4, 1001.8, 1005.2]]),   ##
...            np.array([[1.3, 2.3, 1., 5.9],                 ### y_3 data
...                      [15.4, 14., 12.6, 15.8],             ###
...                      [98.9, 103.2, 100.5, 104.2],         ###
...                      [1000.9, 1003.8, 1002.6, 1006.6]]),  ###
...            np.array([[0.9, 4., 4.9, 0.6],                 #### y_4 data
...                      [11.4, 12.4, 11.7, 14.7],            ####
...                      [98.4, 103.4, 104.3, 104.9],         ####
...                      [1006.3, 1003., 1003.4, 1002.8]]),   ####
...            np.array([[2., 0.5, 5.9, 3.1],                 ##### y_5 data
...                      [11.7, 16.4, 15.7, 14.9],            #####
...                      [104.2, 102.3, 105., 104.4],         #####
...                      [999.6, 1003.3, 1005.3, 1003.7]])]   #####
```

Recall that each array in the list `dataset` represents the exports (in millions of dollars) for a given year, where rows represent countries and columns represent products. We would therefore have:

- $e^{y_1}_{c_1p_1} = \$1.2M$ = `dataset[0][0][0] * 10**6`
- $e^{y_2}_{c_3p_2} = \$102.9M$ = `dataset[1][2][1] * 10**6`

- $e^{y_4}_{c_2 p_1} = \$11.4\text{M}$ = `dataset[3][1][0] * 10**6`

Now let's see how we can convert the `dataset` with a matrix per year into a `timeseries_dataset` with one matrix per country. In order to do the necessary reshaping we simply do:

```
>>> from bipcpg.utils.utils import reshape_year_matrices_to_time_series_matrices
... timeseries_dataset = reshape_year_matrices_to_time_series_matrices(dataset)
```

Note that *reshape_year_matrices_to_time_series_matrices()* converts this into a list of **country** matrices, i.e. the rows of the matrices in `dataset`, not the columns. We therefore get:

```
>>> timeseries_dataset
[array([[1.2,  3. , 1. , 5.4],
         [0.1, 5.2, 4.5, 4.2],
         [1.3, 2.3, 1. , 5.9],
         [0.9, 4. , 4.9, 0.6],
         [2. , 0.5, 5.9, 3.1]]),
array([[10.2,  8.8, 11.7, 15.2],
         [ 9.1, 12.2, 13.4, 11.7],
         [15.4, 14. , 12.6, 15.8],
         [11.4, 12.4, 11.7, 14.7],
         [11.7, 16.4, 15.7, 14.9]]),
array([[101.7,  99.7, 104.2, 103.8],
         [105.5, 102.9, 106.5, 101.9],
         [ 98.9, 103.2, 100.5, 104.2],
         [ 98.4, 103.4, 104.3, 104.9],
         [104.2, 102.3, 105. , 104.4]]),
array([[1001.9, 1002.7, 1000.7, 1004.7],
         [1004.4,  999.4, 1001.8, 1005.2],
         [1000.9, 1003.8, 1002.6, 1006.6],
         [1006.3, 1003. , 1003.4, 1002.8],
         [ 999.6, 1003.3, 1005.3, 1003.7]])]
```

We now have each matrix in the list `timeseries_dataset` representing a country with the export time series as its columns. This is the desired format any dataset should have in order to apply the biPCPG algorithm.

3.2 Computing the average correlation matrix

The input to the PCPG algorithm, which is the last step in the biPCPG algorithm, is a correlation matrix. However, a bipartite dataset consists of a *collection* of multiple samples of data (in our toy example above, multiple countries each exporting multiple products), so the application of the PCPG algorithm to this dataset is not straightforward. To circumvent this problem, the approach taken in the biPCPG algorithm is to compute a correlation matrix for each country and then take the element-wise average of these matrices. This yields a single average correlation matrix which can then be used as the input to the PCPG algorithm.

In order to do this using the `bipcpg` package, we simply take the dataset in a format like `timeseries_dataset`, this is a collection of matrices with observations (which form time series in our example) along its columns and do the following

```
>>> from bipcpg.correlations import get_correlation_matrices_for_list_of_matrices
... correlation_matrices = get_correlation_matrices_for_list_of_matrices(timeseries_
↪dataset)
... avg_correlation_matrix = np.nanmean(correlation_matrices, axis=0)
```

```
>>> avg_correlation_matrix
array([[ 1.      , -0.29375 ,  0.11955 , -0.093725],
       [-0.29375 ,  1.      ,  0.252425, -0.0146  ],
       [ 0.11955 ,  0.252425,  1.      , -0.474325],
       [-0.093725, -0.0146  , -0.474325,  1.      ]])
```

as expect from the linearity of the time series in `timeseries_dataset`, correlation coefficients are all equal to one. It is important to note that *get_correlation_matrices_for_list_of_matrices()* computes the correlations among the **columns** of the matrices in the input list. Also, to filter the returned correlation matrices based on a statistical T-test, we can pass the desired `critical_value` for the p-values, for example `0.05`, as an argument like this:

```
>>> filtered_correlation_matrices = get_correlation_matrices_for_list_of_
↪matrices(timeseries_dataset,
...                                                                                  ␣
↪critical_value=0.05)
```

```
>>> filtered_correlation_matrices
[array([[ 1.      , -0.979757,       nan,       nan],
        [-0.979757,  1.      ,       nan,       nan],
        [      nan,       nan,  1.      ,       nan],
        [      nan,       nan,       nan,  1.      ]]),
array([[ 1., nan, nan, nan],
        [nan,  1., nan, nan],
        [nan, nan,  1., nan],
        [nan, nan, nan,  1.]]),
array([[ 1., nan, nan, nan],
        [nan,  1., nan, nan],
        [nan, nan,  1., nan],
        [nan, nan, nan,  1.]]),
array([[ 1., nan, nan, nan],
        [nan,  1., nan, nan],
        [nan, nan,  1., nan],
        [nan, nan, nan,  1.]])]
```

These `np.nan` values are the result of the filtering of non-statistically significant correlations. This is expected given the very small sample size in our toy dataset.

3.3 Computing the PCPG network

Once we have a correlation matrix, or in the example above, an average correlation matrix `avg_correlation_matrix` we can begin to compute the PCPG network. To do this, first instantiate the PCPG class passing the correlation matrix as an argument

```
>>> from bipcpg.pcpg import PCPG
... pcpg = PCPG(avg_correlation_matrix)
```

we then compute the *average influence* (see *Theory* section) values among the variables in the system

```
>>> pcpg.compute_avg_influence_matrix()
```

```
>>> pcpg.avg_influence_matrix
array([[        nan, -0.01044544, -0.02817951,  0.01193706],
       [-0.04052413,         nan, -0.03887709,  0.01047045],
       [-0.00396688, -0.04729008,         nan, -0.0946936 ],
       [ 0.0182888 , -0.01188309,  0.00370091,         nan]])
```

After computing the avg_influence_matrix we are able to generate the a networkx.DiGraph object of our PCPG network by doing:

```
>>> pcpg.create_network()
```

```
>>> pcpg.network
<networkx.classes.digraph.DiGraph object at 0x7f9bc5559f10>
```

We can check which edges have been included in pcpg.network using networkx:

```
>>> pcpg.network.edges()
OutEdgeView([(0, 1), (1, 3), (1, 2), (2, 0), (3, 0), (3, 2)])
```

or directly via the class attribute edges:

```
>>> pcpg.edges
[(3, 0), (1, 3), (3, 2), (2, 0), (0, 1), (1, 2)]
```

3.4 Computing edge bootstrap values

In order to assess the reliability of a PCPG network's edges we can perform a bootstrap procedure on the dataset timeseries_dataset. As detailed above in *Dataset structure*, this should be an iterable containing matrices whose columns contain observations for one of the the two sets of variables in a bipartite dataset with a matrix for each variable in the complementary set of variables.

To obtain a pandas.DataFrame containing the edge bootstrap values we simply have to do

```
>>> from bipcpg.bootstrap import get_bootstrap_values
... bootstrap_values = get_bootstrap_values(timeseries_dataset, num_replicates=1000)
```

where num_replicates is the number of replicates to be generated in the bootstrap procedure. As when computing correlations for the average correlation matrix (see *Computing the average correlation matrix*). This gives the following results, which may vary when repeated as the bootstrap procedure involves a *random* resampling of the rows in each matrix in timeseries_dataset:

```
>>> bootstrap_values
       0      1      2      3
0  0.000  0.897  0.222  0.288
1  0.099  0.000  0.660  0.606
2  0.774  0.315  0.000  0.264
3  0.708  0.377  0.721  0.000
```

bootstrap_values is a pandas.DataFrame containing the bootstrap values of the *directed* edges in the PCPG network. For a given entry in this dataframe, the row index is the edge's source and the column index is the edge's target. In our example the entry bootstrap_values.loc[2, 0] = 0.774 is the bootstrap value of the edge from product p_3 to product p_1. Note the bootstrap_values dataframe includes the bootstrap values for all *potential* edges in a

PCPG network generated from the `timeseries_dataset`. However, the `pcpg.network` found above will contain only a part of these.

Also note that `critical_value` argument could also be passed to *get_bootstrap_values()* which would filter correlations based on a T-test as described in *Computing the average correlation matrix*.

Note `bootstrap_values` is a `pandas.DataFrame` containing the bootstrap values of the *directed* edges in the PCPG network. For a given entry in this dataframe, the row index is the edge's source and the column index is the edge's target.

These bootstrap values could be added as an attribute to `pcpg.network` we obtained previously by doing:

```
>>> pcpg.add_edge_attribute(attr_data=bootstrap_values, attr_name='bootstrap_value')
```

and we can check the attributes that edges have:

```
>>> import networkx as nx
... nx.get_edge_attributes(pcpg.network, 'bootstrap_value')
{(0, 1): 0.897, (1, 3): 0.606, (1, 2): 0.66, (2, 0): 0.774, (3, 0): 0.708, (3, 2): 0.721}
```

Tip: We recommend reproducing this tutorial's code snippets also including the product names `['p1', 'p2', 'p3', 'p4']` as an argument `variable_names` to *PCPG*, which changes the `pcpg.edges` and `pcpg.nodes` names. We should also pass the same argument to *get_bootstrap_values()* in order to obtain a `bootstrap_values` dataframe with product names as row and column indices.

CHAPTER

FOUR

CODE DOCUMENTATION

4.1 PCPG class

class `bipcpg.pcpg.`**`PCPG`**(*corr_matrix*, *variable_names=None*)

Bases: `object`

Class to obtain a Partial Correlation Planar Graph (PCPG) network from a correlation matrix.[1]

Parameters

- **`corr_matrix`** (*`pandas.DataFrame/numpy.ndarray`*) – Correlation matrix displaying correlations among variables in the system.
- **`variable_names`** (*`list`*) – Names of the variables in the system. The order of this list should coincide with the order of rows and columns in `corr_matrix`.

This class includes methods to perform the necessary computations and obtain a `networkx.Graph` network object. The PCPG algorithm consists in the following steps:

1. Find the *Average influence* (AI) between every *ordered* pair of variables in the system, i.e. those in the input `corr_matrix`. See *`compute_avg_influence_matrix()`*.
2. List the AIs in order from largest to smallest, and,
3. Iterate through the list and add a *directed* edge corresponding to the pair of variables of the AI value in that position **if and only if** (i) the reversed edge is not already in the network and (ii) the network's planarity is not broken by adding the edge. See *`create_network()`*.

See the tutorial for further information.

Variables

- **`avg_influence_matrix`** – `numpy.ndarray` containing average influence values between pairs of variables.
- **`avg_influence_df`** – `pandas.DataFrame` containing average influence values between pairs of variables.
- **`influence_df`** – `pandas.DataFrame` containing influence values between pairs of variables.
- **`partial_corr_df`** – Multi-index `pandas.DataFrame` containing partial correlation values between triple of variables.
- **`network`** – the PCPG network generated (a `networkx.DiGraph` directed graph object).
- **`nodes`** – Nodes in `network`.

[1] Kenett DY, Tumminello M, Madi A, Gur-Gershgoren G, Mantegna RN, Ben-Jacob E (2010) Dominating Clasp of the Financial Sector Revealed by Partial Correlation Analysis of the Stock Market. PLoS ONE 5(12): e15032. <https://doi.org/10.1371/journal.pone.0015032>

- **edges** – Edges in `network`.
- **dict_var_names** – `dict` containing variable numbers as keys and variables names as values.

References

add_edge_attribute(*attr_data*, *attr_name*)
Adds data as an attribute to edges in `network`.

Parameters

- **attr_data** (*dict/pandas.DataFrame*) – `pandas.DataFrame` or `dict` containing edge attribute values.
- **attr_name** (*str*) – Name of attribute to be added to edges.

Note: If `attr_data` is a `pandas.DataFrame`, the row indices should the origin nodes and column indices should be the target nodes. If `attr_data` is a dictionary, keys should be tuples of the form (origin_node, target_node).

add_node_attribute(*attr_data*, *attr_name*)
Adds data as an attribute to nodes in `network`.

Parameters

- **attr_data** (*dict/pandas.Series*) – `pandas.Series` or `dict` containing node attribute values.
- **attr_name** (*str*) – Name of attribute added.

Note: If `edge_attribute_values` is a `pandas.Series`, its index should contain the node and its values the node data. If `edge_attribute_values` is a `dict`, keys should be nodes and values should be node data.

compute_assortativity(*node_attribute*, *attr_type*)
Compute node assortativity based on `node_attribute` of nodes.

Parameters

- **node_attribute** (*str*) – Name of node attribute in `network` by which to compute assortativity.
- **attr_type** (*str*) – Either “qual” or “quant”. Indicates if `node_attribute` data is a qualitative characteristic or a quantitative characteristic.

Returns Value of calculated assortativity.

Return type float

compute_avg_influence_matrix()
Compute average influences between every pair of variables in the system and put these in `avg_influence_matrix`.

Returns None

compute_influence_avg_influence_partial_corr_dfs()
Compute partial correlations, influences and average influences between all variables in the system and put these in `partial_corr_df`, `influence_df` and `avg_influence_df` respectively.

Returns None

`create_network()`
Create PCPG a `networkx.DiGraph` object with `nodes`: and edges found following the PCPG algorithm.

Returns None

`find_edges()`
Compute the edges in the PCPG network using the average influences in `avg_influence_matrix`.

Returns List of edges in the PCPG network

Return type list

4.2 Correlations Functions

`bipcpg.correlations.compute_corr_matrix`(*matrix*, *critical_value=None*)
Obtain a correlation matrix among the variables in a matrix. If `critical value` is passed, the correlation matrix is filtered based on a statistical significance T-test where `critical_value` is the threshold value.

Parameters

- **matrix** (*numpy.ndarray*) – `numpy.ndarray` containing time series for the values of interest with observations along axis 0 (rows) and variables along axis 1 (columns).
- **critical_value** (*float*) – Boundary of the acceptance region of the T-test performed.

Returns Correlation matrix displaying correlation coefficients between the columns (axis 1) of each input matrix.

Return type numpy.ndarray

`bipcpg.correlations.corr_pvalue_matrices`(*matrix*)
Obtain a correlation matrix and p-value matrix for a matrix containing variables and observations.

Parameters **matrix** (*numpy.ndarray*) – 2-dimensional numpy.ndarray containing containing observations axis 0 and variables along axis 1.

Returns tuple containing correlation matrix showing correlation coefficients between columns of input matrix and p-value matrix showing statistical significance of correlations.

Return type tuple

`bipcpg.correlations.get_correlation_matrices_for_list_of_matrices`(*matrices*, *critical_value=None*)
Obtain a correlation matrix and p-value matrix for each matrix (containing variables along the columns and observations along the rows) in `matrices`. If `critical value` is passed, each correlation matrix is filtered based on a statistical significance T-test where `critical_value` is the threshold value.

Parameters

- **matrices** (*Iterable*) – Iterable object containing of 2-dimensional `numpy.ndarray` s with observations along axis 0 (rows) and variables along axis 1 (columns).
- **critical_value** (*float*) – Boundary of the acceptance region of the T-test performed.

Returns list of length len(list_time_series_matrices) containing correlation matrices displaying the correlation coefficients between the columns (axis 1) of each input matrix

Return type list

4.3 Bootstrap functions

`bipcpg.bootstrap.construct_corr_matrix_replicates_from_time_series_matrices(array_of_matrices, num_replicates, critical_value=None)`

Performs a bootstrap procedure on time series matrices to obtain correlation matrix replicates. If `critical_value` is not None, the correlation matrices are filtered using a statistical significance T-test.

Parameters

- **array_of_matrices** (*numpy.ndarray*) – 3-dimensional `numpy.ndarray` with axis 0 representing elements of one of the sets in the bipartite system, axis 1 representing time series observations and axis 2 representing elements of the remaining set in the bipartite system.
- **num_replicates** (*int*) – Number of correlation matrix replicates to be constructed.
- **critical_value** (*float*) – If passed, boundary of the acceptance region of the T-test performed.

Returns Array containing mean of correlation matrix replicates in each batch.

Return type `numpy.ndarray`

`bipcpg.bootstrap.get_bootstrap_values(timeseries_matrices, variable_names=None, num_replicates=1000, critical_value=None)`

Compute bootstrap values for edges in a PCPG network. This function takes a dataset in the form of a list or numpy array of matrices with time series in its columns (see *Dataset structure*) performs a bootstrap procedure that generates a total of `num_replicates` replicate PCPG matrices and finds the bootstrap value of each edge, i.e. the fraction of times the edge appears in these networks. If `critical_value` is not None, the replicate correlation matrices generated are filtered using a statistical significance T-test.

Parameters

- **timeseries_matrices** (*list/numpy.ndarray*) – Iterable containing the dataset for which the PCPG network was generated. This should be a list containing 2d- :class:numpy.ndarray` s whose columns contain observations for one of the the two sets of variables in a bipartite dataset.
- **variable_names** (*list*) – Names of variables along columns of each matrix in `timeseries_matrices`
- **num_replicates** (*int*) – Number of replicates to generate in the bootstrap procedure.
- **critical_value** (*float*) – If passed, boundary of the acceptance region of the T-test performed.

Returns `pandas.DataFrame` containing the bootstrap values of the *directed* edges in the PCPG network. Note that the source of an edge is its row index and the target of the edge is its column index.

Return type `pandas.DataFrame`

4.4 Util functions

bipcpg.utils.utils.**get_degrees_df**(*G*)

Get a `pandas.DataFrame` containing the degree, in-degree and out-degree information of the nodes in `G`.

Parameters `G` (*networkx.DiGraph*) – Directed network.

Returns `pandas.DataFrame` containing degree information.

Return type `pandas.DataFrame`

bipcpg.utils.utils.**remove_reversed_duplicates**(*iterable*)

For an iterable object containing other iterables, yield items which do not have a reversed duplicate in a position with a smaller index.

Parameters **iterable** (*Iterable*) – An iterable object containing other iterables.

Returns Inner iterables which do not have a reversed duplicate in a position with a smaller index.

Return type Iterator[Iterable]

bipcpg.utils.utils.**reshape_year_matrices_to_time_series_matrices**(*list_yearly_matrices*)

For a list of `numpy.ndarray` s, switch the first dimension (list entries) for the second dimension (axis 0) of matrices in the list.

Parameters **list_yearly_matrices** (*list*) – list of 2-dimensional `numpy.ndarray` s indexed over time. Each matrix has one set of variables of the bipartite dataset along axis 0 (rows) and the other set of variables in the bipartite dataset along axis 1 (columns).

Returns list of 2-dimensional `numpy.ndarray` indexed over the elements in the rows of the matrices in `list_yearly_matrices`. Axis 0 (rows) of each matrix is now indexed over time, i.e. the dimension of the elements in `list_yearly_matrices`.

Return type `list`

Example This can be used transform a list of matrices (one per year) into a list of time series matrices. Say we have a list `my_list` containing matrices (one per year) with the exports every country (rows) made for every product (columns). We can then transform this into a list of matrices (one per country) with time series observations along the rows and products along the columns.

```
>>> my_list = [np.array([[1,2],[3,4]]),
...            np.array([[5,6],[7,8]]),
...            np.array([[9,10],[11,12]])]
>>> my_list_transformed = transform_year_matrices_to_time_series_matrices(my_list)
my_list_transformed
[
array([[ 1,  2],
       [ 5,  6],
       [ 9, 10]]),
array([[ 3,  4],
       [ 7,  8],
       [11, 12]])
]
```

bipcpg.utils.utils.**transform_3level_nested_dict_into_df**(*nested_dict*)

Transform a nested dictionary with three levels into a stacked `pandas.DataFrame` with a 2 level multi-index.

Parameters **nested_dict** (*dict*) – Three level nested dictionary to be transformed.

Returns `pandas.DataFrame` with 2-level multi-index. multi-index level 0 corresponds to outermost `nested_dict` keys, multi-index level 1 corresponds to `nested_dict` middle level keys and columns correspond to `nested_dict` innermost keys.

Return type `pandas.DataFrame`

`bipcpg.utils.utils.`**`transform_3level_nested_dict_into_stacked_df`**(*nested_dict*, *name=None*)

Transform a nested dictionary with three levels into a stacked `pandas.DataFrame` with a 3 level multi-index and a single column. If `name` is passed, set the name of the column to `name`.

Parameters

- **`nested_dict`** (*`dict`*) – Three level nested dictionary to be transformed.
- **`name`** (*`str`*) – Name of single column found in returned `pandas.DataFrame`

Returns Stacked dataframe with multi-index level 0 corresponding to outermost nested_dict keys, multi-index level 1 corresponding to nested_dict middle level keys and multi-index level 2 corresponding to nested_dict innermost keys.

Return type `pandas.DataFrame`

`bipcpg.utils.communities_utils.`**`communities_data`**(*G*, ***la_kwds*)

Perform a community detection procedure on graph `G` and return relevant results for plotting.

Parameters

- **`G`** (*`networkx.Graph`*) – `networkx` graph on which to perform community detection.
- **`la_kwds`** – keyword arguments passed on to `leidenalg.find_partition()`.

Returns

- `G_igraph` `igraph.Graph` - *igraph* graph object equivalent to `G`.
- `partition` `leidenalg.VertexPartition` - Graph partition.
- `tup_nodes_num_nodes` `tuple` - a *tuple* containing list of nodes sorted by community and list of number of nodes per community.

Return type `tuple`

`bipcpg.utils.communities_utils.`**`get_igraph_network_and_partition`**(*G*, ***la_kwds*)

Obtain an `igraph` graph and a partition from a `networkx` graph.

Parameters

- **`G`** (*`networkx.Graph`*) – `networkx` graph to be converted into *igraph* graph.
- **`la_kwds`** – keyword arguments passed on to `leidenalg.find_partition()`.

Returns

- `H` `igraph.Graph` - `igraph` graph object.
- `partition` `leidenalg.VertexPartition` - Graph partition.

Return type `tuple`

CHAPTER

FIVE

THEORY

5.1 Partial Correlation Planar Algorithm

The Partial Correlation Planar Graph (PCPG)[1] is based on *partial correlation* which measures the effect that a random variable Z has on the correlation between two other random variables X and Y. The partial correlation is defined in terms of the Pearson correlations $\rho(\cdot,\cdot)$ between the three variables as

$$\rho(X, Y : Z) = \frac{\rho(X,Y) - \rho(X,Z)\rho(Y,Z)}{\sqrt{[1-\rho^2(X,Z)]\,[1-\rho^2(Y,Z)]}}.$$

A small value of $\rho(X, Y : Z)$ may be ambiguous, as this could be due to the correlations among the three variables being small; or due to variable Z having a strong effect on the correlation between X and Y, which is generally the interesting case. In order to discriminate between these two cases the *correlation influence* or *influence* of variable Z on the pair of elements X and Y is used. This is defined as

$$d(X, Y : Z) \equiv \rho(X, Y) - \rho(X, Y : Z).$$

Finally, the metric on which the PCPG is built is the *average influence* of variable Z on the correlations between X and all other variables in the system. This is given by

$$d(X : Z) = \langle d(X, Y : Z)\rangle_{Y \neq X}.$$

An important detail is that, in general, $d(X : Z) \neq d(Z : X)$. The largest among these two quantities indicates the main direction of influence between X and Z, as influence is generally bidirectional. The difference between these two values are often small, which makes a bootstrap procedure necessary in order to asses the confidence in the direction of the average influence, as well as the average influence values.

The construction algorithm of a PCPG network starts with a list of the $N(N-1)$ average influence values in decreasing order and an empty graph of N nodes and no edges, where N is the number of variables in the system. We then cycle through the sorted list, starting with the largest average influence value found, e.g. $d(J : I)$. The edge $I \rightarrow J$ is included in the network if and only if the resulting network is still planar and the edge $J \rightarrow I$ has not been included already.

We stop adding edges if adding the next edge in the list would break the planarity of the graph. This procedure ensures two things: (i) only the largest among $d(X : Z)$ and $d(Z : X)$ will be included in the network, and (ii) the final network has $3(N-2)$ edges. The end result of this procedure is what we refer to as the PCPG network, G.

Naturally, we also obtain the average influence d associated to each edge in G, as well as the network's adjacency matrix $\mathbf{A}$ defined as

$$A_{I,J} = \begin{cases} 1 & \text{if edge } I \rightarrow J \in G, \\ 0 & \text{otherwise.} \end{cases}$$

[1] Kenett DY, Tumminello M, Madi A, Gur-Gershgoren G, Mantegna RN, Ben-Jacob E (2010) Dominating Clasp of the Financial Sector Revealed by Partial Correlation Analysis of the Stock Market. PLoS ONE 5(12): e15032. <https://doi.org/10.1371/journal.pone.0015032>

5.2 References

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

b

INDEX